Finding Attribute-Aware Similar Region for Data Analysis

Feng, Kaiyu; Cong, Gao; Jensen, Christian S.; Guo, Tao

Published in:
Proceedings of the VLDB Endowment

DOI (link to publication from Publisher):
10.14778/3342263.3342277

Publication date:
2019

Document Version
Publisher's PDF, also known as Version of record

Finding Attribute-aware Similar Regions for Data Analysis

Kaiyu Feng[1] Gao Cong[1] Christian S. Jensen[2] Tao Guo[3]
[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[2]Department of Computer Science, Aalborg University, Denmark
[3]Google, Singapore
kfeng002@e.ntu.edu.sg, gaocong@ntu.edu.sg, csj@cs.aau.dk, darkgt@google.com

ABSTRACT

With the proliferation of mobile devices and location-based services, increasingly massive volumes of geo-tagged data are becoming available. This data typically also contains non-location information. We study how to use such information to characterize a region and then how to find a region of the same size and with the most similar characteristics. This functionality enables a user to identify regions that share characteristics with a user-supplied region that the user is familiar with and likes. More specifically, we formalize and study a new problem called the attribute-aware similar region search (*ASRS*) problem. We first define so-called composite aggregators that are able to express aspects of interest in terms of the information associated with a user-supplied region. When applied to a region, an aggregator captures the region's relevant characteristics. Next, given a query region and a composite aggregator, we propose a novel algorithm called *DS-Search* to find the most similar region of the same size. Unlike any previous work on region search, *DS-Search* repeatedly discretizes and splits regions until an split region either satisfies a drop condition or it is guaranteed to not contribute to the result. In addition, we extend *DS-Search* to solve the *ASRS* problem approximately. Finally, we report on extensive empirical studies that offer insight into the efficiency and effectiveness of the paper's proposals.

PVLDB Reference Format:
Kaiyu Feng, Gao Cong, Christian S. Jensen, and Tao Guo.. Finding attribute-aware similar region for data analysis. *PVLDB*, 12(11): 1414-1426, 2019.
DOI: https://doi.org/10.14778/3342263.3342277

1. INTRODUCTION

With the proliferation of location-based services, increasingly massive volumes of spatial objects, often called points of interest (POI), are being accumulated. These include geo-tagged tweets, reviews, business directory listings and tourist information. For instance, as of March 31, 2016, 2.8 million local businesses had been claimed on *Yelp* [8]. These spatial objects often have additional attributes that describe their properties. For example, in real estate listings, apartments for sale have attributes such as price, size, number of bedrooms, and year of construction.

Proceedings of the VLDB Endowment, Vol. 12, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3342263.3342277

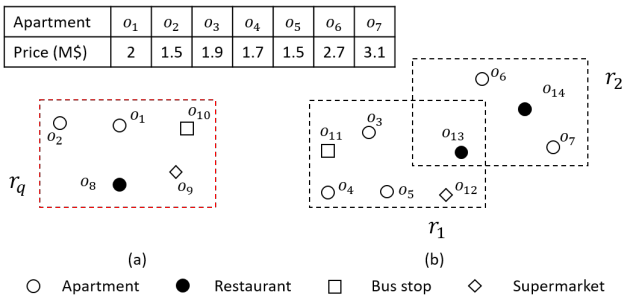


Figure 1: Motivating example.

The availability of massive volumes of spatial objects enables data-driven solutions to real-life problems, similar-region search being one such problem. In real-world settings, many scenarios exist where users may want to search for regions with characteristics similar to those of identified regions. For example, a tourist who is happy with a particular shopping district in an unfamiliar city may want to find another region that is similar to the shopping district. Or a small business owner who is looking to expand may want to find a region with surroundings that are similar to those of the current business, as this may make it possible to reuse existing strategies and concepts for the new business. The following more elaborate example serves to further motivate and illustrate the similar region search problem.

EXAMPLE 1. *Suppose that a user wants to buy an apartment. The ideal neighborhood should satisfy three conditions: (1) The neighborhood should include a restaurant, a supermarket, and a bus stop, which makes it easy to get food and use public transportation. The number of restaurants, supermarkets, and bus stops should not be too large, to avoid noisy neighborhood. (2) The average sales price of apartments in the neighborhood in the past year should be within the user's budget. (3) The neighborhood should be within a region of certain size so that the facilities are within walking distance.*

The user has started in a new job in a new city. How can the user find a region with the above characteristics in this city?

When buying a new apartment, it is a tedious task to find a desirable new region manually, particularly when many factors are to be taken into account. To support applications like this, we provide general functionality that, given a region and a description of the characteristics to be considered, finds a similar region.

In order to find a similar region, two important questions must be answered: (1) *What does a region look like?* and (2) *How is the similarity between two regions to be defined?* To answer the first question, we observe that the displays of smartphones, laptops,

tablets, and navigation devices generally, are rectangular. So we contend that defining a region to be rectangular is a natural choice that makes it easy for users to specify regions. This idea is also adopted in previous studies [5, 11, 24].

To answer the second question, we notice that different users are likely to be interested in different aspects of a region. In Example 1, the user is interested in two aspects: the presence of different categories of POIs in the region, specifically restaurants, supermarkets, and bus stops, and the average sales price of apartments in the region. These two aspects share a common characteristic. They are both aggregates of attribute values associated with spatial objects in a region. Specifically, we compute the number of POIs of each category for all the POIs in the region, and we compute the average sales price of the apartments in the region. In order to support a broad class of applications, we aim to support the specification of a broad class of notions of similarity. To do that, we introduce the notion of a composite aggregator. Users can use their own composite aggregators to define the aspects that they are interested in. The result of applying a composite aggregator to a region is a feature vector that describes the region w.r.t. the aspects specified in the aggregator. This then enables computation of the similarity between two regions.

More specifically, we generalize the problem described in Example 1 and formulate the *attribute-aware similar region search* (*ASRS*) problem. Let F be a composite aggregator that defines the aspects that a user is interested in. Given a region r_q of size $a \times b$, the *ASRS* problem is to find a similar region r of size $a \times b$ such that the distance between the aggregate representations $F(r)$ and $F(r_q)$ of the two regions is minimal.

We propose a novel exact algorithm called *DS-Search* that solves the *ASRS* problem. Unlike all previous work on region search [5, 11, 12, 21] that is based on the sweep line algorithm, *DS-Search* adopts a different and novel tack. We first reduce the *ASRS* problem to the attribute-aware similar point (*ASP*) problem. The reduction enables us to find a point from $O(n^2)$ disjoint regions, where n is the number of spatial objects, instead of from an infinite set of points in space. To solve the *ASP* problem, we first use a grid to discretize the space into cells, which are classified into two groups: clean cells and dirty cells. We compute an intermediate result by processing the clean cells and estimate lower bounds of the distance for the dirty cells. The dirty cells whose lower bounds exceed the intermediate result can be safely pruned. Then we split the space containing the remaining dirty cells into two smaller sub-spaces. We repeatedly apply this discretize-split procedure to the smaller sub-spaces until either the space satisfies a drop-condition or all dirty cells in the grid are pruned. When the algorithm terminates, it returns the exact answer to the *ASRS* problem. The complexity of *DS-Search* is $O(\Omega n)$, where n is the number of spatial objects, and parameter Ω depends only on the GPS accuracy and is independent of the number of spatial objects. Hence, we can treat Ω as a fixed constant. When the number of spatial objects is much larger than Ω, the complexity of *DS-Search* can then be viewed as $O(n)$, which is much better than the $O(n^2)$ complexity of the sweep line algorithm. In our experimental study, *DS-Search* is 2–3 orders of magnitude faster than the sweep line algorithm. Moreover, we introduce indexing along with pruning techniques to improve the efficiency of *DS-Search*.

Since a slight imprecision may be desirable in some applications, if this substantially reduces the processing time, we extend *DS-Search* to solve the *ASRS* problem approximately. This extension of *DS-Search* finds a region that is very similar to the query region with a better efficiency.

In summary, the key contributions are as follows:

(1) To the best of our knowledge, this is the first study of the attribute-aware similar region search (*ASRS*) problem.

(2) We develop a novel algorithm called *DS-Search* to solve the *ASRS* problem. Its time complexity is $O(\Omega n)$, where n is the number of spatial objects, and Ω is a constant determined by the accuracy of GPS tracking devices.

(3) We propose a novel index structure to improve the efficiency of *DS-Search*. We further extend *DS-Search* to solve the *ASRS* problem approximately with a better efficiency.

(4) By applying implementations of the proposed algorithms to both real-life and synthetic datasets, we experimentally investigate their efficiency and effectiveness in finding the most similar region. The results show that *DS-Search* is 2–3 orders of magnitude faster than a baseline algorithm, which is adapted from the sweep line algorithm [11, 21]. We also show that with a slight modification, our algorithm is about one order of magnitude faster than the sweep line algorithm [21] for the *MaxRS* problem.

2. RELATED WORK

Range Aggregate Query. The range aggregate query [4, 15, 16, 22] takes as input a query range q and a set O of spatial objects, and it outputs the total weight of the spatial objects in the query range. In contrast, the *ASRS* problem aims to find a range of a given size such that it is most similar to the query region. These two problems have different settings. Hence, the approaches designed for range queries cannot be applied to solve the *ASRS* problem.

Spatial Keyword Query. Different types of spatial keyword queries (SKQ) typically take a location and a set of keywords as input. Early studies [3, 7, 9, 10, 14, 18] aim to find a set of geo-textual objects such that each object is relevant to the query keywords and is close to the query location. Instead of retrieving single objects, several more recent studies retrieves groups of objects. One kind of such functionality is the notion of collective spatial keyword queries (CoSKQ). For example, the mCK query [6, 13, 25] aims to find a group of objects covering all the query keywords while minimizing the distances between the objects in the group. Cao et al. [2] study five types of CoSKQ that use different objective functions. Similar to the CoSKQ, given a source and a destination, the optimal route query [17, 19, 23] aims to find the shortest route from the source to the destination that covers POIs belonging to POI categories (e.g., a gas station or a post office) specified in the query.

The above types of spatial keyword queries are fundamentally different from the *ASRS* problem. Specifically, the *ASRS* problem computes a more general aggregate representation when capturing a region's characteristics. Therefore, our problem applies to a broader range of spatial data, including, but not limited to, geo-textual data. Moreover, the *ASRS* problem aims to find a rectangular region, not a set of objects. Consequently, new techniques are needed to solve the *ASRS* problem.

Region Search Problem. Our problem is closely related to the region search problem. A class of studies [5, 11, 21, 24] aim to find a region of a given size such that the aggregate score of the region is maximized. For instance, the *max-enclosing rectangle* (*MER*) problem [21] aims to find a rectangular region of a given size that encloses the maximum number of objects. This problem is further refined as the *maximizing range sum* (*MaxRS*) problem [5, 24]. Next, Feng et al. [11] study the best region search (*BRS*) problem that extends the aggregate function from SUM to support submodular monotone functions. Also, Mostafiz et al. [20] extend the MaxRS problem by taking the types of the spatial objects into account. Specifically, they apply constraints to the types of the spatial objects while searching for the region with the maximum total weight.

The *ASRS* problem is different from the aforementioned region search problems that do not take a query region as input, as does *ASRS*. The *MER* problem [21] and the *MaxRS* [5] problem use SUM as the aggregate function to search for a region, while the *ASRS* problem considers different types of attributes and uses the distance between the feature vectors of a candidate region r and the query region. Hence, the *ASRS* problem is more general, and the *MER* and *MaxRS* problems are special cases of the *ASRS* problem. In our experimental study, we adapt our proposed algorithm to solve the *MaxRS* problem and find that our algorithm is about one order of magnitude faster than the state-of-the-art algorithm for *MaxRS* [5, 21, 24].

Furthermore, we propose a novel algorithm, *DS-Search*, whose time complexity is $O(\Omega n)$. This algorithm is fundamentally different from the sweep line algorithm used in all previous work [5, 11, 21, 24]. The time complexity of the sweep line based algorithms is dependent on the score function used: when using a simple score function like SUM, the time complexity is $O(n \log n)$ [5, 21], where n is the number of spatial objects. When using a submodular function, the time complexity is $O(n^2)$ [11]. When applying the sweep line algorithm to solve the *ASRS* problem, its time complexity is $O(n^2)$, to be discussed in Section 4.1.

3. PROBLEM STATEMENT

We proceed to formulate the ***A**ttribute-aware **S**imilar **R**egion **S**each* (*ASRS*) problem.

3.1 Terminology

We denote the set of attributes considered by $\mathcal{A} = \{A_1, \ldots, A_m\}$, and we denote the domain of attribute A_i by $dom(A_i)$.

Next, we let O denote the set of *spatial objects* considered. For a spatial object o, we use $o.\rho$ to denote its geo-location, and $o[A_i]$ to denote its value of attribute A_i. For example, a set of POIs can be viewed as a set of spatial objects. Attribute set $\mathcal{A}$ may contain attributes like "category," "rating," and "price." The domain of attribute "category" may contain values such as "restaurant." As another example, geo-tagged tweets can be viewed as spatial objects. We can assign a topic to each tweet according to its textual content by adopting an existing topic model [1]. Then the attribute set $\mathcal{A}$ may contain attributes like "hashtag" and "topic." The domain of attribute "topic" may contain pre-defined topics.

3.2 Composite Aggregator

There is no single right way of defining the similarity between two regions. Rather, different users may be concerned with different aspects of a region. For instance, one user may think two regions are similar because they both contain many apartments, while another user may think that the regions are different because one contains old and inexpensive apartments while the other contains new and expensive apartments. How can a user express the aspects of interest of a region, and how can we capture the characteristics of the region w.r.t. those aspects? Next, we introduce the notion of a **composite aggregator**, which enables users to define the aspects that they are interested in. The result of applying a composite aggregator to a region is a feature vector that describes the region w.r.t. the aspects specified in the aggregator.

We start with the notion of **aggregator**, which computes a feature vector for a region w.r.t. a given attribute.

DEFINITION 1. **Aggregator.** *An aggregator f takes as input a region r, an attribute A, and a selection function γ that selects a set $\gamma(r)$ of objects from region r that satisfy certain conditions. Aggregator f computes a feature vector from the set $\gamma(r)$ of objects w.r.t. the attribute A.*

We consider three kinds of aggregators. Each aggregator takes a region r, an attribute A, and a selection function γ as input.

(1) Distribution Aggregator f_D: Aggregator f_D computes the distribution of objects in $\gamma(r)$ according to their values of attribute A. The distribution is represented as a d-dimensional vector, where $d = |dom(A)|$ is the count of all possible values for attribute A. The i-th dimension of the vector is given as follow: $f_D(r, A, \gamma)[i] = |\{o | o \in \gamma(r) \wedge o[A] = a_i\}|$.

(2) Average Aggregator f_A: Aggregator f_A computes the average value of attribute A for all objects in $\gamma(r)$, i.e., $f_A(r, A, \gamma) = \frac{1}{|\gamma(r)|} \sum_{o \in \gamma(r)} o[A]$.

(3) Sum Aggregator f_S: Aggregator f_S computes the sum of the values of the attribute A for all objects in $\gamma(r)$, i.e., $f_S(r, A, \gamma) = \sum_{o \in \gamma(r)} o[A]$.

EXAMPLE 2. *We illustrate the outputs of the three aggregators by using the example in Fig 1. The domain of "category" is $dom(category) = \{$Apartment, Supermarket, Restaurant, Bus stop$\}$. Let γ_{all} be a selection function that selects all objects. Aggregator $f_D(r_q, category, \gamma_{all}) = (2, 1, 1, 1)$ computes a 4-dimensional vector that captures the category distribution of the objects in r_q (two apartments, one supermarket, one restaurant, and one bus stop). Let γ_{apt} be a selection function that selects the objects whose "category" is "Apartment." Aggregator $f_A(r_q, Price, \gamma_{apt}) = \frac{2+1.5}{2} = 1.75$ computes the average sales price of the apartments in r_q. Aggregator $f_S(r_q, Price, \gamma_{apt}) = 2 + 1.5 = 3.5$ computes the sum of sales prices for the apartments in r_q.*

Remark. We consider three useful aggregators. For instance, we can use f_D to compute the categorical distribution of POIs in a region, which reflects the functionality of the region. A region with many apartments is very likely a residential area.Next we can use f_A to compute the average price of apartments for sale in a region, which reflects an important characteristic of a region; we can use f_S to compute the total number of residents, which is useful in city planing tasks, like allocating medical resources. We emphasize the proposed solution is not limited to these three aggregators. Users can define their own aggregators to support various applications.

So far, we have made it possible to use an aggregator $f \in \{f_D, f_A, f_S\}$ together with an attribute A and a selection function γ to define an aspect of interest. Next, we introduce the notion of a composite aggregator F that makes it possible to define multiple aspects of interest.

DEFINITION 2. **Composite aggregator.** *A composite aggregator is defined as a k tuple $F = ((f_1, A_1, \gamma_1), \ldots, (f_k, A_k, \gamma_k))$, where $f_i \in \{f_D, f_A, f_S\}$ is an aggregator, A_i is an attribute, and γ_i is a selection function, $i \in [1, k]$.*

We next define the aggregate representation as follows.

DEFINITION 3. **Aggregate Representation.** *When applied to a region r, a composite aggregator $F = ((f_1, A_1, \gamma_1), \ldots, (f_k, A_k, \gamma_K))$ computes a vector $F(r)$ that is the concatenation of the outputs of $f_i(r, A_i, \gamma_i)$, $i \in [1, k]$. We refer to $F(r)$ as the aggregate representation of r w.r.t. composite aggregator F.*

EXAMPLE 3. *In Example 2, let $F = ((f_D, Category, \gamma_{all}), (f_A, Price, \gamma_{apt}))$ be a composite aggregator. The representation of r_q w.r.t. F is $F(r_q) = (2, 1, 1, 1, 1.75)$.*

The aggregate representation captures a region's characteristics w.r.t. the aspects defined in the composite aggregator. This then enables the computation of the relevant similarity between two regions.

3.3 Problem Definition

We are now ready to define the *ASRS* problem.

DEFINITION 4. **Attribute-aware Similar Region Search (*ASRS*) problem.** *Given a set O of spatial objects, a query region r_q of size $a \times b$, and a composite aggregator F, the attribute-aware similar region search problem aims to find a region r of size $a \times b$ such that*

$$r = \arg\min_{r} dist(F(r), F(r_q)),$$

where $dist(F(r), F(r_q))$ is the distance between the representations of r and r_q, which is defined as follows.

$$dist(F(r), F(r_q)) = \sum |F(r)[i] - F(r_q)[i]| \cdot \mathbf{w}[i],$$

where $\mathbf{w}$ is a weight vector that specifies the user's preference for each dimension in the representation.

The *ASRS* problem follows the idea of *query by example*. The query region can be a real region in space. For instance, user may want to find a region in city A with the same function as a region in city B. The query region can also be a virtual region handcrafted by a user to describe his interests.

Note that the distance is based on the L1-norm. Our proposals can be modified easily to use other distance metrics, e.g., L2-norm. For ease of presentation, we only cover the L1-norm distance.

EXAMPLE 4. *In Example 2, consider a weight vector $\mathbf{w} = (1, 1, 1, 1, 1)$. The* ASRS *problem aims to find the most similar region for r_q w.r.t. the composite aggregator F. We have $F(r_1) = (3, 1, 1, 1, 1.6)$ and $F(r_2) = (2, 0, 2, 0, 2.9)$ for regions r_1 and r_2, respectively. Since the representation of r_q is $F(r_q) = (2, 1, 1, 1, 1.75)$, we have $dist(F(r_q), F(r_1)) = 1.15$, and $dist(F(r_q), F(r_2)) = 4.15$. Region r_1 is thus more similar to r_q than r_2.*

4. EXACT SOLUTION

To obtain an exact solution to the *ASRS* problem, we first introduce the attribute-aware similar point (*ASP*) problem and can reduce the *ASRS* problem to the *ASP* problem. Next, we present a novel algorithm, *DS-Search*, that solves the *ASP* problem (and thus the *ASRS* problem) efficiently.

4.1 The *ASP* Problem

The most similar region can be at any location in space, which is infinite. Hence, it is prohibitively expensive to consider every possible location. This challenge is also faced by existing studies of region search problems [5, 11, 21]. Inspired by the idea of transforming the *MaxRS* problem to the rectangle intersection problem [21], we reduce the *ASRS* problem to the *ASP* problem. We first define terminology to be used later.

DEFINITION 5. **Rectangle.** *A rectangle object $\mathfrak{r}$ is a rectangle of size $a \times b$ that is associated with a set of attributes. We use $\mathfrak{r}.\rho$ to denote the location of the rectangle's top-right corner. We use $\mathfrak{r}[A_i]$ to denote its value for attribute A_i.*

Given a location p in a space, we say a rectangle object $\mathfrak{r}$ covers p if p is inside $\mathfrak{r}$. Let $\mathcal{R}_p$ be the set of rectangle objects that cover p. Consider the example in Fig 2(b). Location p is covered by two rectangles, i.e., $\mathcal{R}_p = \{\mathfrak{r}_5, \mathfrak{r}_6\}$. Recall that in the *ASRS* problem, we apply a composite aggregator to a region r to compute a representation from the set of spatial objects in r. Similarly, we can apply a composite aggregator to a location p and compute a representation of p from $\mathcal{R}_p$. With a slight abuse of notation, we use $F(p)$ to denote the aggregate representation computed from $\mathcal{R}_p$ and refer to this as p's **aggregate representation**. In Fig 3(b), by applying $F = ((f_D, \text{Color}, \gamma_{all}))$ to p, the aggregate representation is $(1, 1)$, as p is covered by one red and one blue rectangle. We refer to the distance between representations $F(p)$ and $F(r_q)$ as p's **distance**.

We can now define of the *ASP* problem.

DEFINITION 6. **Attribute-aware Similar Point (*ASP*) problem.** *Given a set $\mathcal{R}$ of rectangle objects, a composite aggregator F, and a query representation $F(r_q)$, the* ASP *problem aims to find a location p that minimizes $dist(F(p), F(r_q))$.*

The *ASRS* problem can be reduced to the *ASP* problem as follows. For each object o_i in the *ASRS* problem, we generate a rectangle object of size $a \times b$ whose top-right corner is located at o_i. We use the same composite aggregator and query representation in the *ASP* problem as in the *ASRS* problem. Therefore, we get an instance of the *ASP* problem. We illustrate the reduction with the example in Fig 2. In the *ASRS* problem, each spatial object is associated with the attribute "color." To reduce the *ASRS* problem to the *ASP* problem, for each spatial object o_i, $i \in [1, 6]$, we generate a rectangle of size $a \times b$ with the same attribute and whose top-right corner is located at o_i, as depicted in Fig 2(b). Note that when reducing *ASRS* to *ASP*, we can also generate a rectangle in other ways, such as making the spatial object any of the four corners of the rectangle or the centroid of the rectangle. We simply use top-right corner to illustrate the idea of reduction.

Next, we prove that we can solve the *ASRS* problem by solving the reduced *ASP* problem, and we justify the reduction by presenting a property of the *ASP* problem.

LEMMA 1. *Consider a location p from the* ASP *problem and the rectangular region r whose bottom-left corner is located at p in the* ASRS *problem. A rectangle object $\mathfrak{r}_i$ covers p iff the corresponding spatial object o_i is inside r.*

PROOF. Since a rectangle object $\mathfrak{r}_i$ is drawn by making the corresponding spatial object o_i the top-right corner, we have $\mathfrak{r}_i.\rho.x = o_i.x$, $\mathfrak{r}_i.\rho.y = o_i.y$. If a spatial object o_i is inside r, we have $p.x < o_i.x < p.x + a$, $p.y < o_i.y < p.y + b$. We can derive $\mathfrak{r}_i.\rho.x - a < p.x < \mathfrak{r}_i.\rho.x$, $\mathfrak{r}_i.\rho.y - b < p.y < \mathfrak{r}_i.\rho.y$, i.e., $\mathfrak{r}_i$ covers p. Similarly, we can get o_i is inside r if $\mathfrak{r}_i$ covers p. □

EXAMPLE 5. *In Fig 2, location p is made the bottom-left corner of the dashed rectangular region r. In the* ASRS *problem, the dashed region r encloses o_5 and o_6, while location p is covered by $\mathfrak{r}_5$ and $\mathfrak{r}_6$ in the* ASP *problem.*

THEOREM 1. *Consider an instance of the* ASRS *problem. Let p be the answer to the* ASP *problem reduced from the* ASRS *problem. Then the rectangular region r of size $a \times b$ whose bottom-left corner is located at p is an answer to the* ASRS *problem.*

PROOF. Let p be a location in space in the reduced *ASP* problem, and r be the corresponding rectangular region of size $a \times b$ in the *ASRS* problem, whose bottom-left corner is located at p. According to Lemma 1, the spatial objects in r in the *ASRS* problem have the same attributes as the rectangles that cover p in the *ASP* problem. Since we use the same composite aggregator and query representation, the region r in the *ASRS* problem has the same distance as the location p in the *ASP* problem. If p in the *ASP* problem has the minimum distance then r also has the minimum distance and is an answer to the *ASRS* problem instance. □

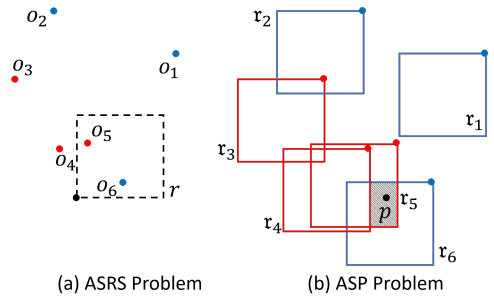


Figure 2: An example of reduction. The composite aggregator $F = ((f_D, \text{Color}, \gamma_{all}))$ computes the distribution of objects according to their colors. The query representation is $(\#red, \#blue) = (1, 1)$. The weight vector for computing distance between two representations is $\mathbf{w} = (1, 1)$.

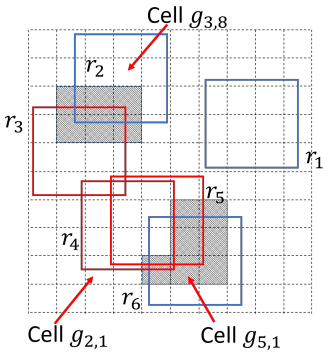


Figure 3: Discretization.

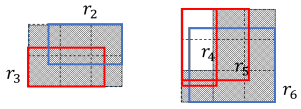


Figure 4: Two regions after splitting.

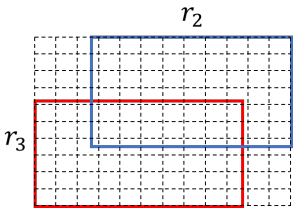


Figure 5: A region satisfying the drop condition.

EXAMPLE 6. *In Fig 2, an ASRS problem is reduced to an ASP problem. In the reduced ASP problem, location p is an answer, as it is covered by exactly one red and one blue rectangle. In the ASRS problem, the dashed region whose bottom-left corner is located at p is an answer, as it covers one red and one blue spatial object.*

Due to Theorem 1, we can solve *ASRS* by solving the *ASP* problem. We next show a property of the *ASP* problem that justifies the reduction.

In the *ASP* problem, the edges of the rectangles divide the space into many **disjoint regions**. Consider the example in Fig 2(b). The grey region, which belongs to the overlap of r_5 and r_6, is a disjoint region. We have the following property of disjoint regions.

LEMMA 2. *Any location in a disjoint region is covered by the same set of rectangles.*

In Fig 2(b), any location in the grey disjoint region is covered by $\{r_5, r_6\}$. This implies that we only need to find the disjoint region, whose covering rectangles have the aggregate representation that is most similar to the query representation.

LEMMA 3. *There are $O(n^2)$ disjoint regions, where n is the number of rectangles [21].*

Consequently, by reducing *ASRS* to *ASP*, we convert our problem from selecting a region from an infinite set to *selecting a disjoint region from a set of $O(n^2)$ disjoint regions*.

It is then natural to ask the following question: *How can we find the disjoint region with the minimum distance in the set of $O(n^2)$ disjoint regions?* One idea is to adapt the sweep line algorithm as in previous work [5,11,12,21] to scan the space and check all disjoint regions. Specifically, we use a sweep line to scan the space. During the sweeping, the sweep line is divided into intervals by the edges of the rectangles. For each interval, we maintain a distance based on the overlapping rectangles. A point from the interval with the minimum distance during the sweeping is the solution to the *ASP* problem. The time complexity of this solution is $O(n^2)$, where n is the number of rectangle objects in the *ASP* problem.

Having quadratic complexity, the efficiency of this solution degrades rapidly as the number of rectangles increases. To address this challenge, we propose the novel **D**iscretize and **S**plit search (*DS-Search*) algorithm.

4.2 Overview of DS-Search

Instead of moving a sweep line to locate every disjoint region and examine its distance, *DS-Search* incorporates a new idea to tackle *ASP*. Specifically, *DS-Search* discretizes the space into cells with a grid. The cells can be classified into clean cells and dirty cells (to be discussed in Section 4.3). We process the clean cells to get an intermediate result. We also estimate a distance lower bound for each dirty cell. The dirty cells whose lower bounds exceed the intermediate result can be safely pruned. The remaining dirty cells are split into two groups, each of which corresponds to a sub-space smaller than the original space. Each of the two smaller sub-spaces are then discretized again. We then check the clean and dirty cells in the smaller space to refine the intermediate result. We repeat this step until either there is no dirty cell whose lower bound is smaller than the current result or the space satisfies the drop condition (Section 4.5). When the algorithm terminates, the intermediate result is an exact answer to *ASP*. Due to these ideas, *DS-Search* has better time complexity and practical performance than the sweep line solution.

DS-Search has three key procedures: discretizing the space, splitting the space, and checking the drop condition. We next elaborate on each of these in turn with the running example of the reduced *ASP* in Fig 2(b).

4.3 Discretizing the Space

We first discretize the space under consideration into $n_{row} \times n_{col}$ cells with a grid. Here n_{row} and n_{col} are pre-specified parameters. We use $g_{i,j}$ to denote the cell in the i-th column and j-th row. For instance, the example in Fig 2(b) is discretized into 10×10 cells, as depicted in Fig 3.

For each rectangle, we can find the sets of cells that it fully covers and partially covers, respectively. For instance, cell $g_{3,8}$ is fully covered by r_2, and $g_{2,1}$ is partially covered by r_4. We can classify the cells into two categories according to their relationship with the rectangles: (1) **Clean cell:** A cell is clean if no rectangle partially covers it, and (2) **Dirty cell:** A cell is dirty if at least one rectangle partially covers it.

In Fig 3, $g_{3,8}$ is a clean cell. Cell $g_{5,1}$ is a dirty cell, as r_4 and r_5 partially cover it.

The clean and dirty cells are processed separately.

Processing clean cells. According to the definition, a clean cell is fully inside a disjoint region. As a result, any location p in a clean cell g is covered by the same set of rectangles, denoted by $\mathcal{R}_g$. Hence, we can apply the composite aggregator to $\mathcal{R}_g$ to get its aggregate representation and then compute the distance to the query representation. In this case, the disjoint region that contains this clean cell is examined. We take any location from the clean cell with the minimum distance as an intermediate result. Note that

all locations in this clean cell have the same distance. We simply take the center of the cell as the intermediate result.

Processing dirty cells. As there are rectangles that partially cover a dirty cell, different locations inside a dirty cell may have different distances. To prune the dirty cells, we propose to estimate a lower bound of distance for the locations inside a dirty cell.

To estimate a lower bound of distance, we first need to compute two vectors $\overline{\mathbf{v}}$ and $\underline{\mathbf{v}}$ to bound the aggregate representation $\mathbf{v}$ of any location in the dirty cell, i.e., $\underline{\mathbf{v}}[i] \leq \mathbf{v}[i] \leq \overline{\mathbf{v}}[i]$ for any $0 \leq i \leq d-1$. Once we have $\underline{\mathbf{v}}$ and $\overline{\mathbf{v}}$, the lower bound of distance can be computed as follows:

$$lb = \sum w[i] \cdot \begin{cases} F(r_q)[i] - \overline{\mathbf{v}}[i] & \text{if } F(r_q)[i] > \overline{\mathbf{v}}[i] \\ \underline{\mathbf{v}}[i] - F(r_q)[i] & \text{if } F(r_q)[i] < \underline{\mathbf{v}}[i] \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

LEMMA 4. *For any location p in a dirty cell g, let $\mathbf{v}$ be its aggregate representation. We have $lb < dist(\mathbf{v}, F(r_q))$ as long as $\underline{\mathbf{v}}[i] \leq \mathbf{v}[i] \leq \overline{\mathbf{v}}[i]$ holds for any $i \in [0, d-1]$.*

PROOF. If $F(r_q)[i] > \overline{\mathbf{v}}[i]$, we have $|F(r_q)[i] - \mathbf{v}[i]| \geq |F(r_q)[i] - \overline{\mathbf{v}}[i]|$. If $F(r_q)[i] < \underline{\mathbf{v}}[i]$, we have $|F(r_q)[i] - \mathbf{v}[i]| \geq |\underline{\mathbf{v}}[i] - F(r_q)[i]|$. If $\underline{\mathbf{v}}[i] \leq F(r_q)[i] \leq \overline{\mathbf{v}}[i]$, we have $|F(r_q)[i] - \mathbf{v}[i]| \geq 0$. Putting these together, we get $lb \leq dist(F(r_q), \mathbf{v})$. □

We next elaborate on how to compute $\underline{\mathbf{v}}$ and $\overline{\mathbf{v}}$ to bound the aggregate representation $\mathbf{v}$ of any location in a dirty cell.

For a dirty cell g, let $\mathcal{R}_g^f$ and $\mathcal{R}_g^p$ be the sets of rectangles that fully and partially cover g, respectively. Let $\overline{\mathcal{R}}_g = \mathcal{R}_g^f \cup \mathcal{R}_g^p$, $\underline{\mathcal{R}}_g = \mathcal{R}_g^f$. For any location p in g, let $\mathcal{R}_p$ be the set of rectangles that cover p. We observe that $\underline{\mathcal{R}}_g \subseteq \mathcal{R}_p \subseteq \overline{\mathcal{R}}_g$. This motivates us to bound the aggregate representation by utilizing $\underline{\mathcal{R}}_g$ and $\overline{\mathcal{R}}_g$.

Recall that the aggregate representation is the concatenation of the outputs of aggregators. We next present how to bound the output of an aggregator by using $\overline{\mathcal{R}}_g$ and $\underline{\mathcal{R}}_g$. For brevity, we consider the distribution aggregator f_D as an example. We can bound the output of other aggregators similarly.

The i-th dimension of the output of f_D is the number of rectangles in $\mathcal{R}_p$ having a_i as the value for attribute A. We can compute the bounding vectors as follows

$$\underline{\mathbf{v}}_D[i] = |\{\mathfrak{r} | \mathfrak{r} \in \underline{\mathcal{R}}_g \wedge \mathfrak{r}[A] = a_i\}|$$

$$\overline{\mathbf{v}}_D[i] = |\{\mathfrak{r} | \mathfrak{r} \in \overline{\mathcal{R}}_g \wedge \mathfrak{r}[A] = a_i\}|$$

LEMMA 5. *Let $\mathbf{v}_D$ be the output of f_D. We have $\underline{\mathbf{v}}_D[i] \leq \mathbf{v}_D[i] \leq \overline{\mathbf{v}}_D[i]$, $i \in [0, d-1]$.*

PROOF. Since $\underline{\mathcal{R}}_g \subseteq \mathcal{R}_p \subseteq \overline{\mathcal{R}}_g$, for each value a_i of attribute A, we have $\{\mathfrak{r} | \mathfrak{r} \in \underline{\mathcal{R}}_g \wedge \mathfrak{r}[A] = a_i\} \subseteq \{\mathfrak{r} | \mathfrak{r} \in \mathcal{R}_g \wedge \mathfrak{r}[A] = a_i\} \subseteq \{\mathfrak{r} | \mathfrak{r} \in \overline{\mathcal{R}}_g \wedge \mathfrak{r}[A] = a_i\}$. Hence, we have $\underline{\mathbf{v}}_D[i] \leq \mathbf{v}_D[i] \leq \overline{\mathbf{v}}_D[i]$ for any $i \in [0, d-1]$. □

EXAMPLE 7. *In the example in Fig 3, cell $g_{2,1}$ is partially covered by $\mathfrak{r}_4$ and $\mathfrak{r}_5$. The aggregate representation of p is bounded by $\overline{\mathbf{v}} = (2, 0)$ and $\underline{\mathbf{v}} = (0, 0)$. According to Equation 1, the lower bound of p's distance is $lb = 0 + 1 = 1$.*

Cell $g_{5,1}$ is fully covered by $\mathfrak{r}_6$ and partially covered by $\mathfrak{r}_4$ and $\mathfrak{r}_5$. The aggregate representation of p is bounded by $\overline{\mathbf{v}} = (2, 1)$ and $\underline{\mathbf{v}} = (0, 1)$. According to Equation 1, the lower bound of p's distance is $lb = 0 + 0 = 0$.

With the lower bounds estimated, we can safely prune the dirty cells whose lower bounds exceed that of the intermediate result.

Function Discretize(c, p_{opt}, d_{opt})

```
1  Construct a grid with n_row × n_col cells.;
2  foreach rectangle r in space do
3  |  Mark the cells that are fully and partially covered;
4  foreach cell g do
5  |  if g is a clean cell then
6  |  |  v_g ← the aggregate representation of g ;
7  |  |  d_g ← the distance between v_g and the query
   |  |    representation;
8  |  |  if d_g < d_opt then d_opt ← d_g, p_opt ← center of g;
9  |  else
10 |  |  g.lb ← Lower bound of distance for any location in g;
```

The discretization procedure is outlined in Function Discretize. The function takes as input the space c to be processed and two variables p_{opt} and d_{opt}, which are used to store the location with the minimum distance. It first constructs a grid with $n_{row} \times n_{col}$ cells (line 1). Then, for each rectangle $\mathfrak{r}$, it marks the cells that $\mathfrak{r}$ fully and partially cover (lines 2–3). Next, it iterates through every cell in the grid (lines 5–10). If the cell is clean, it computes the aggregate representation and the distance to the query representation (lines 5–7). Otherwise, it computes the distance lower bound for the locations in the cell (line 10). We take the center of the clean cell with the minimum distance as an intermediate result (line 8).

EXAMPLE 8. *Consider Fig 3. When invoking Function Discretize, we compute the distance to the query representation for each clean cell. Cell $g_{3,8}$ is a clean cell covered by $\mathfrak{r}_2$. The aggregate representation for a point at any location in $g_{3,8}$ is $(0, 1)$, and the distance is $1 + 0 = 1$. Since $g_{3,8}$ has the current minimum distance, we update p_{opt} with the center of $g_{3,8}$ and set d_{opt} to 1. The lower bounds of all dirty cells are also computed.*

4.4 Splitting the Space

In the previous step, we pruned the dirty cells whose lower bounds exceed that of the current result. Here, we present how to deal with the remaining dirty cells.

We propose to split the space containing the remaining dirty cells into two smaller sub-spaces, and we then invoke Function Discretize again to discretize each sub-space. Since the sizes of the sub-spaces are smaller, the sizes of the cells become smaller, and we can estimate tighter bounds for the dirty cells, making them more likely to be pruned. By repeatedly splitting and discretizing, we can prune more dirty cells while gradually improving the intermediate result.

The high level idea of splitting is as follows: We first partition the remaining dirty cells into two groups. For each group, we return the minimum bounding rectangle (MBR) that encloses all dirty cells in the group as a new space.

When we partition the dirty cells, we have three goals: (1) We aim to minimize the total area of the two MBRs for the two groups. (2) The overlap between the two MBRs should be minimized. (3) The numbers of the rectangles that overlap with the MBRs should be balanced.

Sometimes the three goals conflict, making it impossible to optimize all three at the same time. We adopt a two-step heuristic algorithm to address the problem: First, we choose two seed sets of dirty cells such that we can gradually expand the two sets to complete the partitioning. Specifically, we select two dirty cells that are furthest from each other as the initial seed sets. Second, we expand the two seed sets by adding the remaining cell to the sets gradually. For each dirty cell, we compute the cost of adding it to a set. The cost is defined as the increase of the area of the MBR when adding

Function Split(G, d_{opt})

1 $G_{dirty} \leftarrow$ dirty cells whose lower bounds are smaller than d_{opt};
2 $g_1, g_2 \leftarrow$two cells from G_{dirty} that are far from each other;
3 $G_1 = \{g_1\}, G_2 = \{g_2\}$;
4 **for** each $g \in G_{dirty} \setminus \{g_1, g_2\}$ **do**
5 $\quad cost_1 = area(MBR(G_1 \cup \{g\})) - area(MBR(G_1))$;
6 $\quad cost_2 = area(MBR(G_2 \cup \{g\}) - area(MBR(G_2))$;
7 $\quad$ **if** $cost_1 > cost_2$ **then** $G_2 = G_2 \cup \{g\}$;
8 $\quad$ **else** $G_1 = G_1 \cup \{g\}$;
9 $lb_1 \leftarrow \min_{g \in G_1} lowerbound(g)$;
10 $lb_2 \leftarrow \min_{g \in G_2} lowerbound(g)$;
11 **return** $MBR(G_1), lb_1, MBR(G_2), lb_2$

it to the group. The dirty cell is added to the set with the minimum cost.

The pseudocode of the split procedure is given in Function Split. It takes as input the grid G that is used to discretize the space, and the current minimum distance d_{opt}. We first select the dirty cells from G whose lower bounds are smaller than d_{opt}(line 1). Next, we select two cells from G_{dirty} that are farthest from each other and initialize the two seed sets to be g_1 and g_2 (lines 2–3). Then, for each remaining cell g, we compute the costs of adding it to the two seed sets (lines 5–6). We add cell g to the set where the cost is smallest (lines 7–8). When all cells in G_{dirty} have been processed, we compute the smallest lower bound for each set (lines 9–10) and return the MBRs that enclose the cells in the two sets together with the lower bounds (line 11).

EXAMPLE 9. *In Fig 3, the grey cells are the dirty cells whose lower bounds are smaller than the current minimum distance. We select* $\{g_{6,1}\}$ *and* $\{g_{1,7}\}$ *as two seeds of dirty cells. Then we expand the two sets gradually. For instance, the cost of adding* $g_{2,7}$ *to* $\{g_{1,7}\}$ *is* $2 - 1 = 1$*, while the cost of adding* $g_{2,7}$ *to* $\{g_{6,1}\}$ *is* $35 - 1 = 34$*. Thus, we add* $g_{2,7}$ *to set* $\{g_{1,7}\}$*. We repeat this procedure until all the dirty cells whose lower bounds are smaller than the current minimum distance are processed. The two MBRs are shown in Fig 4.*

4.5 Drop Condition

As covered in Section 4.4, we repeatedly split and discretize the space. The question is when can we stop splitting the space. Clearly, if every dirty cell has a lower bound larger than the intermediate result, there is no need to split because all dirty cells can be pruned. We proceed to introduce a drop condition and show that we can safely stop splitting if the space satisfies the drop condition.

We start by defining GPS accuracy, which will be used later.

DEFINITION 7. **GPS horizontal/vertical accuracy.** *Let X and Y be the sets of x-coordinates and y-coordinates of the vertical and horizontal edges of the rectangles in $\mathcal{R}$, respectively. We define the* ***horizontal (vertical) accuracy****, denoted by Δ_X (Δ_Y), as the minimum distance between any two distinct values in X (Y), i.e., $\Delta_X = \min |x_i - x_j|$ for any $x_i, x_j \in X, x_i \neq x_j$ ($\Delta_Y = \min |y_i - y_j|$ for any $y_i, y_j \in Y, y_i \neq y_j$).*

Note that the horizontal and vertical accuracies cannot be infinitely small. They are bounded by the resolution of the positioning techniques and are unrelated to the cardinality of the dataset. Therefore, we treat the GPS horizontal/vertical accuracies as constants.

We next present the **drop condition**.

DEFINITION 8. **Drop condition.** *Given a space that is discretized by a grid, we say that the space* ***satisfies the drop condition*** *if both of the following conditions are satisfied:*

$$2 \cdot w_c < \Delta_X, \quad 2 \cdot h_c < \Delta_Y,$$

where w_c and h_c are the width and height of a cell, and Δ_X and Δ_Y are the horizontal and vertical accuracies.

THEOREM 2. *If a space satisfies the drop condition, every disjoint region in the space contains at least one clean cell.*

PROOF. We prove the theorem by contradiction. Consider a space that satisfies the drop condition, i.e., $2 \cdot w_c < \Delta_X$, $2 \cdot h_c < \Delta_Y$, where w_c and h_c are the width and height of a cell. We assume that a disjoint region exists that encloses no cell. Let x_1 and x_2 (y_1 and y_2) be the x (y) coordinate of any two vertical (horizontal) edges of the disjoint region. Since $|x_2 - x_1| \geq \Delta_X$, and $|y_2 - y_1| \geq \Delta_Y$, we have $|x_2 - x_1| \geq 2 \cdot w_c$, and $|y_2 - y_1| \geq 2 \cdot h_c$. Hence, this disjoint region encloses a rectangular region of size at least $2 \cdot w_c \times 2 \cdot h_c$. Hence, no matter how the position of the grid varies, this disjoint region always encloses at least one cell, which contradicts the assumption. □

EXAMPLE 10. *Consider the space to the left in Fig 4. We discretize it with a* 10×10 *grid, as depicted in Fig 5. The edges of* $\mathfrak{r}_2$ *and* $\mathfrak{r}_3$ *divide the space into three disjoint regions. Every disjoint region encloses at least one clean cell. Hence, this space satisfies the drop condition, and we do not need to split it again.*

4.6 The DS-Search Algorithm

We now have all the machinery in necessary to describe *DS-Search*. The algorithm, shown in Algorithm 1, takes as input the original space c, the current point with the minimum distance p_{opt}, and the current minimum distance d_{opt}. It returns a region of size $a \times b$ with the minimum distance. The algorithm first reduces the *ASRS* problem to the *ASP* problem by generating a rectangles for each spatial object (line 1). Then it uses a heap H to maintain the spaces to be processed. The heap is initialized to $(c, 0)$ (line 2). It processes the spaces in the heap greedily (lines 3–10). In each iteration, it first invokes Function Discretize and updates p_{opt} and d_{opt} by examining the clean cells (line 5). If the space does not satisfy the drop condition, it invokes Function Split to obtain two smaller sub-spaces (line 7). The two smaller sub-spaces are then pushed into the heap (line 8). The algorithm terminates when the lower bounds of the unprocessed spaces in the heap are not smaller than d_{opt}. It returns the region r_{opt} of size $a \times b$ whose bottom-left corner is located at p_{opt} (lines 10–11).

EXAMPLE 11. *We use the reduced ASP problem in Fig 2 to illustrate the DS-Search algorithm. In the first iteration, we discretize the space into a* 10×10 *grid, as shown in Fig 3. We update* p_{opt} *and* d_{opt} *by examining the clean cells. Currently, the minimum distance is 1. Since the space in Fig 3 does not satisfy the drop condition, we split it into two sub-spaces, as shown in Fig 4. The two smaller sub-spaces are pushed onto the heap. In the second iteration, assume that we pop out the left space in Fig 4 from the heap. We discretize it with a* 10×10 *grid again, as shown in Fig 5. We update the* p_{opt} *and* d_{opt} *by examining all clean cells. Currently, the minimum distance is 0. This space satisfies the drop condition, so we do not need to split it again. As the lower bound of the remaining spaces in the heap is equal to 0, the algorithm terminates.*

Algorithm 1: DS-Search Algorithm

Input : Original space c, the current best point p_{opt} and its distance d_{opt}
Output: A region in c with the minimum distance

$\mathcal{R} \leftarrow$ the set of rectangles whose top-right corner is an object and each of which overlaps with c;
$H \leftarrow$a min-heap; Push $(c, 0)$ into H;
repeat
  $c \leftarrow$ the space in H with the minimum lower bound;
  `Discretize(`c`,` p_{opt}`,` d_{opt}`)`;
  if *c does not satisfy the drop condition* **then**
    $c_1, lb_1, c_2, lb_2 \leftarrow$`Split(`$G$`)`;
    Push (c_1, lb_1), (c_2, lb_2) into H;
until $H.top().lb \geq d_{opt}$;
$r_{opt} \leftarrow$ a rectangular region whose bottom-left corner is located at p_{opt};
return r_{opt}

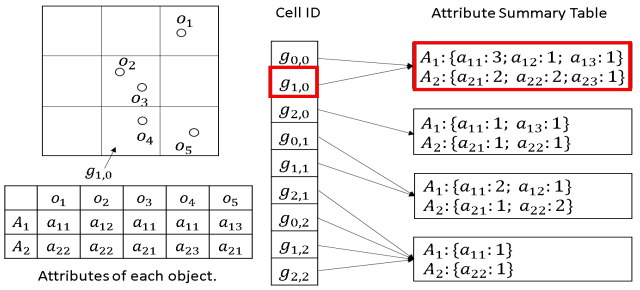
LEMMA 6. *Let $\Omega = \frac{W \cdot H(n_{col}+n_{row})}{\Delta_X \cdot \Delta_Y \cdot n_{col} \cdot n_{row}}$, where W and H are the width and height of the original space, n_{col} and n_{row} are parameters specified by the user, and Δ_X and Δ_Y are the horizontal and vertical accuracies. The complexity of Algorithm 1 is $O(\Omega \cdot n)$, where n is the number of spatial objects.*

PROOF. In Function Discretize, we need to compute the distance for each clean cell and estimate a lower bound for each dirty cell. The time complexity is $O(n \cdot (n_{col} \cdot n_{row}) + n_{col} \cdot n_{row} \cdot d)$, where n is the number of rectangles, n_{col} and n_{row} are the parameters specified by user to control the grid for the discretization, and d is the dimensionality of the aggregate representation. The time complexity of Function Split is $O(n_{row} \times n_{col})$. In *DS-Search*, each time when we discretize a space, we split it into two smaller sub-spaces unless it satisfies the drop condition. The spaces that are processed form a binary tree. The depth of the tree is $O(\log \frac{W}{n_{col} \cdot \Delta_X} + \log \frac{H}{n_{row} \cdot \Delta_Y})$, where W and H are the width and height of the original space, and Δ_X and Δ_Y are the horizontal and vertical accuracies. Therefore, there are $O(\frac{W \cdot H}{n_{col} \cdot n_{row} \cdot \Delta_X \cdot \Delta_Y})$ spaces to be processed. Putting these together, the time complexity of Algorithm 1 is $O(\Omega \cdot n)$. □

Remark. In parameter $\Omega = \frac{W \cdot H(n_{col}+n_{row})}{\Delta_X \cdot \Delta_Y \cdot n_{col} \cdot n_{row}}$, n_{row} and n_{col} are user-specified parameters, and Δ_X and Δ_Y are constants that are only determined by positioning technology used, as discussed in Section 4.5. Moreover, W and H are bounded by the size of the real world and thus can also be viewed as constants. Putting these together, the parameter Ω is independent of the number of spatial objects and is a constant. When the number of spatial objects is much larger than Ω, the complexity of Algorithm 1 can be viewed as being $O(n)$. This makes the algorithm more efficient than the traditional sweep line based approaches. According to our experiments, the *DS-Search* algorithm is 2–3 orders of magnitude faster than the sweep-line based baseline algorithm.

LEMMA 7. *Algorithm 1 computes the exact answer to* ASRS.

PROOF. We denote by d the optimal disjoint region in the reduced *ASP* that has the minimum distance. Let c be a space in heap H that overlaps with this disjoint region. If c does not satisfy the drop condition, we split c into two smaller sub-spaces. At least one sub-space overlaps with disjoint region d. As we repeatedly split and discretize space, the sizes of the cells get smaller and smaller. Eventually, the space will satisfy the drop condition in finite steps.



	o_1	o_2	o_3	o_4	o_5
A_1	a_{11}	a_{12}	a_{11}	a_{11}	a_{13}
A_2	a_{22}	a_{22}	a_{21}	a_{23}	a_{21}

Figure 6: The structure of a grid index.

When the space c satisfies the drop condition, this means at least one clean cell is inside disjoint region d. We can find a location in disjoint region d when we examine all clean cells in c. Hence, Algorithm 1 always returs the exact answer to the *ASRS* problem. □

5. ENHANCING DS-SEARCH WITH A GRID INDEX

We propose a pruning technique based on a grid index to further improve the efficiency of *DS-Search*.

5.1 Overview

The *ASRS* problem has a "locality" property: the aggregate representation of a region is only determined by the set of spatial objects inside the region. Motivated by this observation, we propose a pruning technique by following the divide-and-conquer strategy.

The high-level idea is as follows: We first use a grid index to divide the space into cells. For each cell in the grid, we estimate a distance lower bound for the set of candidate regions whose bottom-left corners are in the cell. Then the cells are searched greedily by invoking *DS-Search*: the cells with smaller lower bounds are searched first. We terminate when the lower bounds of the unsearched cells are not smaller than the current minimum distance.

Note that this idea is different from the discretization of a region as introduced in Section 4.3. There, the discretization is applied to the rectangles generated in the reduced *ASP* problem. The reduction from the *ASRS* problem to the *ASP* problem is query-dependent, i.e., we need the size of the query rectangle to conduct the reduction. Hence, the discretization has to be done during query processing. In contrast, the grid introduced here is used to index the spatial objects. Its granularity is independent of the query, and it is constructed before querying occurs.

5.2 Grid Index

The grid index is essentially a grid consisting of $s_x \times s_y$ cells, where s_x and s_y are pre-specified and are independent from the query. We use $g_{i,j}$ to denote the cell in the i-th column and j-th row. We use $G[^{i_2}_{i_1}][^{j_2}_{j_1}]$ to denote the region consisting of cell $g_{i,j}$ for any $i_1 \leq i < i_2$ and $j_1 \leq j < j_2$.

Each cell in the grid is assigned an **attribute summary table**. This table contains a number of entries of the form $(A_s : T_{A_s})$, where A_s is an attribute, and T_{A_s} is a hash table that maps a value $a_m \in dom(A_s)$ to the number of objects having a_m as the value of attribute A_s. For a cell $g_{i,j}$, its attribute summary table is built over the objects in all cells in $G[^{\infty}_{i}][^{\infty}_{j}]$.

EXAMPLE 12. *Fig 6 illustrates the grid index. Consider the cell $g_{1,0}$. Its attribute summary table, as shown in the figure, is built over the objects in $G[^{\infty}_{1}][^{\infty}_{0}]$, which is the entire set of objects. For instance, three objects have a_{11} as the value of A_1. Thus, the entry*

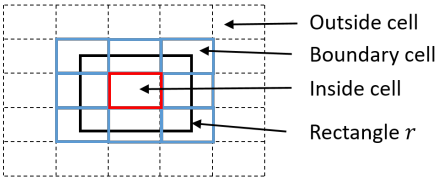


Figure 7: Bounding and bounded regions.

of A_1 in the corresponding hash table contains the key-value pair $a_{11} : 3$.

Note that many cells share the same attribute summary table. In Fig 6, cells $g_{2,1}$, $g_{0,2}$, $g_{1,2}$, and $g_{2,2}$ have the same attribute summary table. Hence, we use a hash map to record a cell and its attribute summary table to avoid redundant storage, as shown in Fig 6.

The grid index has the following property.

LEMMA 8. *Consider a region $G[{}^{r}_{l}][{}^{t}_{b}]$. Let n_{a_j} be the number of objects in the region having a_j as the value of A_i. We can efficiently obtain n_{a_j} by utilizing the grid index as follows:*

$$n_{a_j} = g_{l,b}.T_{A_i}[a_j] + g_{r,t}.T_{A_i}[a_j] - g_{l,t}.T_{A_i}[a_j] - g_{r,b}.T_{A_i}[a_j]$$

According to Lemma 8, we can efficiently compute the number of objects with a particular value of a specified attribute in a region $G[{}^{r}_{l}][{}^{t}_{b}]$.

5.3 Estimating a Lower Bound

We say that a candidate region r is **bl-corner-located in** a cell $g_{i,j}$ if the bottom-left corner of r is in $g_{i,j}$. Next we describe how to estimate a distance lower bound for the candidate regions bl-corner-located in a cell. We first introduce the concepts of **bounding region** and **bounded region**.

DEFINITION 9. **Bounding and bounded region.** *Given a candidate region r and a grid G, a cell is an **inside cell** if it is fully covered by r, a cell is an **outside cell** if it does not overlap with r, and a cell is a **boundary cell** if it is partially covered by r.*

*We define the region consisting of the inside cells as the **bounded region** of r, denoted by $\underline{G}$. We define the region consisting of the inside cells and the boundary cells as the **bounding region** of r, denoted by $\overline{G}$.*

EXAMPLE 13. *In Fig 7, the red cell is an inside cell, the dotted cells are outside cells, and the blue cells are boundary cells.*

For the set of candidate regions that are bl-corner-located in a cell $g_{i,j}$, we can compute the bounding and bounded regions according to the cell size and the size of the candidate region. Given a candidate region r and its bounding and bounded regions $\overline{G}$ and $\underline{G}$, we have $O(\underline{G}) \subseteq O(r) \subseteq O(\overline{G})$, where $O(r)$, $O(\overline{G})$, and $O(\underline{G})$ are the sets of objects in r, $\overline{G}$, and $\underline{G}$, respectively. As a result, we can estimate a lower bound of the distance for r as we did in Section 4.3. Specifically, we first compute two vectors $\overline{\mathbf{v}}$ and $\underline{\mathbf{v}}$ to bound the aggregate representation of $\mathbf{v}$. Then we compute the lower bound based on Equation 1.

5.4 Search Using the Grid Index

Algorithm 2 describes how to utilize the grid index to improve the efficiency of *DS-Search*. It takes as input a grid index $\mathcal{G}$. We use a min-heap H to maintain the cells to be processed (line 1). We use d_{opt} and p_{opt} to maintain the current minimum distance and

Algorithm 2: GI-DS

Input : A grid index $\mathcal{G}$, a composite aggregator F, query region r_q
Output: The most similar region r

1 $H \leftarrow$ a min-heap, $d_{opt} \leftarrow \infty$, $p_{opt} \leftarrow null$;
2 **for** *each cell g in $\mathcal{G}$* **do**
3 $\quad lb \leftarrow$lower bound for the candidate regions in g;
4 $\quad$ Push (lb, g) into heap H;
5 **while** *H is not empty $\wedge$ $H.top.lb < d_{opt}$* **do**
6 $\quad lb, g \leftarrow H.pop()$;
7 $\quad r_{opt} \leftarrow$`DS-Search`$(g, lb, p_{opt}, d_{opt})$;
8 **return** r_{opt};

the point with the minimum distance (line 1). First, for each cell g in the grid index, we compute a lower bound of the distance for the candidate regions that are bl-corner-located in g as explained in Section 5.3 (line 3). The cell g and its lower bound lb are pushed onto the heap (line 4). Then we search the cells iteratively (lines 5–7). In each iteration, we pop out the cell with the minimum lower bound (line 6), and invoke Algorithm 1 to find the region with the minimum distance in g (line 7). We repeat this procedure until the lower bounds of the unsearched cells are not smaller than the current minimum distance d_{opt} (line 5).

6. AN APPROXIMATE SOLUTION

In some applications, a slight imprecision of the result returned may be preferable, if this reduces the processing time substantially. Here, we extend *DS-Search* to solve the *ASRS* problem approximately.

We first define the $(1+\delta)$-approximate *ASRS* problem.

DEFINITION 10. **$(1+\delta)$-approximate *ASRS* problem.** *Given a set O of spatial objects, a query region r_q of size $a \times b$, a composite aggregator F, and a parameter $\delta > 0$, the $(1+\delta)$-approximate* ASRS *problem aims to find a region r of size $a \times b$ such that*

$$dist(F(r), F(r_q)) \leq (1+\delta) dist(F(r_{opt}), F(r_q)),$$

where r_{opt} is the optimal region that has the minimum distance.

The quality of the approximation can be controlled by choosing an appropriate δ. A smaller δ yields a better approximation.

To solve the $(1+\delta)$-approximate *ASRS* problem, we make two major changes to *DS-Search*. The first relates to how we split a region (Function Split in Section 4.4). In the exact solution, we split a region into two smaller regions by partitioning the dirty cells whose lower bounds are smaller than the current minimum distance into two sets and return the two MBRs that enclose the cells in each set. In the approximate solution, it is not necessary to consider all dirty cells. Specifically, we change line 1 in Function Split to let G_{dirty} be the set of dirty cells whose lower bounds are smaller than d_{opt}, where d_{opt} is used to maintain the current minimum distance.

The second major change relates to the use of the grid index that we use to enhance *DS-Search*. In Algorithm 2, we iteratively process each candidate in a greedy manner until the lower bound of the top candidate region in the heap exceeds the current minimum distance (lines 5–7 in Algorithm 2). Instead, we now terminate the process early when the lower bound of the top candidate region exceeds $\frac{d_{opt}}{1+\delta}$, where d_{opt} is used to maintain the current minimum distance.

With these modifications, we have the following theorem.

THEOREM 3. *Let r_{app} be the region found by the modified DS-Search algorithm. We have $dist(F(r_{app}), F(r_q)) \leq (1+\delta) \cdot dist(F(r_{opt}), F(r_q))$, where r_{opt} is the optimal region with the minimum distance.*

PROOF. We can prove by contradictory. Assume that $dist(F(r_{app}), F(r_q)) > (1+\delta) \cdot dist(F(r_{opt}), F(r_q))$. Let r' be a region whose distance is smaller than $\frac{dist(F(r_{app}),F(r_q))}{(1+\delta)}$. Let g be the candidate region where r' is bl-corner-located. Then g's lower bound is smaller than $\frac{dist(F(r_{app}),F(r_q))}{(1+\delta)}$. Hence g will be processed by *DS-Search* in *GI-DS*. Similarly, let c be a cell in *DS-Search* that contains the bottom-left corner of r'. If c is a clean cell, then r' will be returned. If c is a dirty cell, its lower bound is smaller than $\frac{dist(F(r_{app}),F(r_q))}{(1+\delta)}$, indicating that c will be further split. This procedure repeats until c is a clean cell, and r' is then returned. In both cases, r' will be returned, which contradicts the assumption. Hence, the theorem is proved. □

7. EXPERIMENTAL STUDY

We present the setup of our experiments, and then investigate the performance of the proposed algorithms. All algorithms are implemented in C++ and compiled by VS 2015. All experiments are run on a Windows PC with an Intel Xeon 3.70 GHz CPU and 16 GB memory.

7.1 Experimental Setup

Datasets. We use real and synthetic data in the experimental study. We use a real dataset Tweet that consists of 3.2×10^8 geo-tagged tweets posted in the U.S. The dataset was crawled from June 2014 to December 2016. The ranges of latitude and longitude of the tweets are $[24.39, 49.39]$ and $[-124.87, -66.86]$, respectively. Its GPS horizontal and vertical accuracies are both $\Delta X = 10^{-8}$, $\Delta Y = 10^{-8}$.

Next we use a synthetic dataset POISyn that is generated from Tweet. Specifically, for each tweet in Tweet, we generate a spatial object that has the same location as the tweet, and we assign two attributes "Rating" and "Number of visits" to each such object. The "rating" is computed as follows: rating $= \frac{|tweet|}{\max|tweet|} \cdot 10$, where $|tweet|$ is the length of the text content of the corresponding tweet, and $\max|tweet|$ is the maximum length of any tweet in the dataset. Therefore, the domain of "rating," dom(rating) is $[0, 10]$. The "number of visits" of a spatial object is randomly selected from $[1, 500]$.

Composite Aggregator. In order to evaluate our proposed approaches, we design two composite aggregators, one for each dataset. The details of the composite aggregator are as follows:

Composite Aggregator 1: The first composite aggregator is designed for Tweet. Assume we want to find a region that is highly correlated to weekend. Specifically, most of the geo-tagged tweets inside the region are posted during weekends rather than on weekdays. In order to find such a region, we need a composite aggregator that computes the distribution of tweets based on the day of the week they are posted. Hence, we use the following composite aggregator: $F_1 = ((f_D, \text{day of the week}, \gamma_{all}))$. Note that $|dom(\text{day of the week})| = 7$. Thus F_1 outputs a 7-dimensional vector, where the i-th dimension corresponds to the number of tweets posted on the i-th weekday.

We next present how to use a query region to describe our interests. Since we aim to find a region with more tweets on weekends than on weekdays, we consider a query region r_q whose aggregate representation is $F(r_q) = (0, 0, 0, 0, 0, T_6, T_7)$, where T_6 and T_7 are the maximum number of tweets on Saturday and Sunday, respectively, that a region can have. We use a weight vector $\mathbf{w} = (\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{2}, \frac{1}{2})$ when computing the distance between two representations. With the given composite aggregator, query representation, and weight vector, a candidate region is more similar to the query if more tweets occur on weekends and fewer tweets occur on weekdays.

Composite Aggregator 2: The second composite aggregator is designed for POISyn. Assume we want to find a region such that many people visit POIs in the region, and such that the average rating of the POIs is very good. To find such a region, we define a composite aggregator that computes the average rating of the POIs and the sum of number of visits of the POIs: $F_2 = ((f_S, \text{Number of visits}, \gamma_{all}), (f_A, \text{Rating}, \gamma_{all}))$. This composite aggregator computes a 2-dimensional vector.

To describe our interests, we consider a query region r_q whose aggregate representation is $F(r_q) = (v_{max}, 10)$, where v_{max} is the maximum number of visits a region can have. We use weight vector $\mathbf{w} = (\frac{1}{v_{max}}, \frac{1}{10})$ to compute the distance between two representations. With the given composite aggregator, query representation, and weight vector, a candidate region is more similar to the query region if the total number of visits in the region is large and the average rating of POIs in the region is high.

Query Rectangle Size. Let W and H be the width and height of the minimum rectangle that encloses all the spatial objects. We set $q = \frac{W}{1000} \times \frac{H}{1000}$ to be the unit size of a query rectangle, and we define $k \cdot q = (k \cdot \frac{W}{1000}) \times (k \cdot \frac{H}{1000})$. We then vary the size of the query rectangle by using different values for k.

Evaluated algorithms. We evaluate the following algorithms. (a) The *DS-Search* algorithm; (b) The *GI-DS* algorithm. We use *64-GI-DS*, *128-GI-DS*, and *256-GI-DS* to denote the *GI-DS* algorithm with grid index granularity 64×64, 128×128, and 256×256, respectively; (c) The extension of *GI-DS* for the $(1+\delta)$ approximate *ASRS* problem, denoted by *app-GIDS*; and (d) the sweep line based algorithm, denoted as *Base*, which is adapted from the literature [11, 21].

7.2 Performance of DS-Search

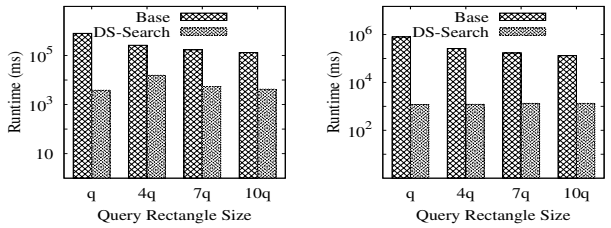
We evaluate the performance of *DS-Search* and compare with *Base*.

Effect of query rectangle size. We first evaluate the effect of the size of the query rectangle. We use four query rectangle sizes: q, $4q$, $7q$, $10q$. We set the parameters $n_{col} = 30$ and $n_{row} = 30$ for *DS-Search*. We extract 1 million objects from Tweet and POISyn to form two new datasets, denoted as Tweet-1M and POISyn-1M, respectively. Fig 8 reports the runtime of *DS-Search* and *Base*. The y-axis uses a logarithmic scale.

We observe that *DS-Search* is orders of magnitude faster than *Base*. This is because the complexity of *Base* is $O(n^2)$, while the complexity of *DS-Search* is close to $O(n)$, where n is the number of spatial objects. We also observe that *DS-Search* is affected less by the size of the query rectangle than *Base*.

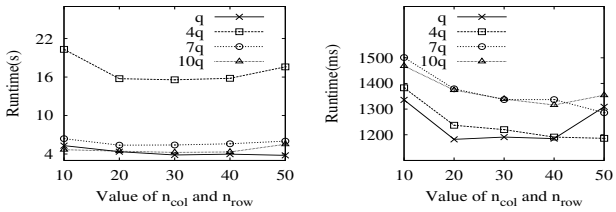
Effect of n_{col} and n_{row}. In this set of experiments, we evaluate the effect of parameters n_{col} and n_{row}. We vary n_{col} and n_{row} to control the granularity of the grid. We use four combinations: 10×10, 20×20, 30×30, 40×40, and 50×50. Fig 9 shows the resulting runtime of *DS-Search*.

We observe that *DS-Search* achieves the best performance when the parameters are set to $n_{col} = n_{row} = 30$. The granularity of the grid has an significant impact on the efficiency. When we use a fine granularity, there is a large number of cells in the grid. Since we need to compute the aggregate representation for each clean cell and to estimate a lower bound for each dirty cell, the time cost



(a) **Tweet-1M** (b) **POISyn-1M**

Figure 8: Runtime v.s. query rectangle size.



(a) **Tweet-1M** (b) **POISyn-1M**

Figure 9: Runtime of *DS-Search* w.r.t. n_{col} and n_{row}.

will be high due to the large number of cells. On the other hand, when we use a coarse granularity, the width and height of each cell is large, making it unlikely to satisfy the drop condition. We need to balance these two factors when we assign values to the two parameters.

Scalability. Finally, we evaluate the scalability of *DS-Search* and *Base* w.r.t. the number of geo-tagged objects in the dataset. We use $10q$ as the query rectangle size. We set the parameters $n_{col} = n_{row} = 30$ for *DS-Search*. Fig 10 depicts the runtime when varying the dataset cardinality. The y-axis uses a logarithmic scale.

We observe that *DS-Search* is about 2–3 orders of magnitude faster than *Base*, especially when the number of objects becomes larger. This occurs because *Base* has a much higher complexity than *DS-Search*.

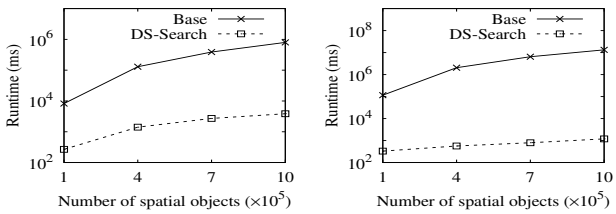
7.3 Performance of the GI-DS Algorithm

We proceed to study the performance of the *GI-DS* algorithm.

Effect of granularity. We first investigate the effect of the granularity of the grid index on the efficiency using three granularities: 64×64, 128×128, and 256×256. We extract 100 million geo-tagged objects from Tweet and POISyn to form datasets Tweet-100M and POISyn-100M, respectively. Fig 11 reports the runtime of the four algorithms w.r.t. the query rectangle size.

We observe that *GI-DS* outperforms *DS-Search* in most cases. In particular, *GI-DS* algorithm achieves the best performance when the granularity is set to 128×128. The running time of *GI-DS* is about 47% of the running time of *DS-Search* on average.

Moreover, we observe that when the granularity of the grid index is too coarse or too fine, the efficiency of *GI-DS* degrades. When the granularity is too coarse, the lower bound estimated for each

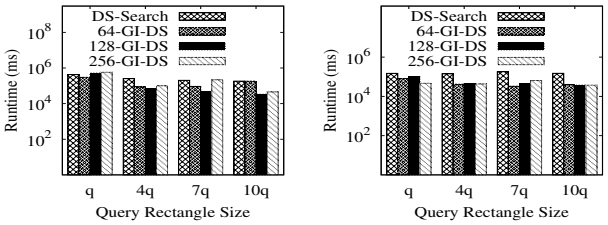


(a) **Tweet** (b) **POISyn**

Figure 10: Runtime v.s. number of objects.

Table 1: Ratio of cells searched on Tweet-100M and index size.

	Query rectangle size				Index size
Granularity	q	$4q$	$7q$	$10q$	
64×64	24.0 %	22.7%	19.2%	13.8%	2.2 MB
128×128	8.1%	7.1%	5.5%	7.6%	8.6 MB
256×256	2.3%	2.0%	1.4%	2.0%	33.6 MB



(a) **Tweet-100M** (b) **POISyn-100M**

Figure 11: Runtime v.s. granularity of grid index.

cell is loose, making it difficult to prune unnecessary cells. On the other hand, when the granularity is too fine, two nearby cells in the grid may be covered by the almost the same set of rectangles. Thus, the two cells have similar lower bounds, meaning that the granularity introduces redundant computations.

Index size. The grid index needs to maintain a pointer from each cell to an attribute summary table. The sizes of grid indices with different granularity are reported in Table 1. We observe that since the granularity of the grid is relatively coarse (from 64×64 to 256×256), the indices take a little space.

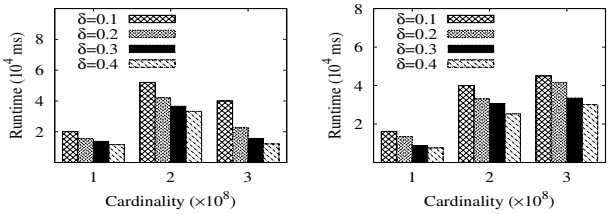
Usefulness of lower bound. To investigate the usefulness of the lower bound estimation in the *GI-DS* algorithm, we run *64-GI-DS*, *128-GI-DS*, and *256-GI-DS* on Tweet-100M and report the ratio of cells in the grid index that are searched by *DS-Search*. The results reported in Table 1 show that only a small fraction of cells are searched. Moreover, the fraction of cells that are searched decreases when the granularity of the grid index increases. This is because we can estimate a much tighter lower bounds when a fine granularity is adopted.

7.4 Performance of the app-GIDS Algorithm

We proceed to evaluate the approximate solution.

Efficiency. In this set of experiments, we use four values for parameter δ: 0.1, 0.2, 0.3 and 0.4. We vary the cardinality of the dataset and report the runtime of the approximate solutions with different δs in Fig 12. We observe that the runtime decreases as δ increases. This is because fewer dirty cells are considered in the phase of splitting the region in *DS-Search* when a large δ is used. As a result, the subregions tend to be smaller and are more likely to satisfy the drop condition, yielding a better efficiency. In addition, more candidate regions in the grid index are pruned when δ is large.

Result quality. We next conduct experiments to evaluate the quality of the regions returned by the approximate solution. The quality

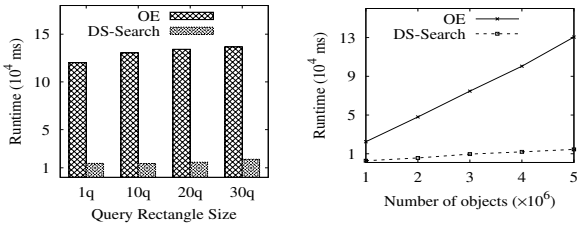


(a) Composite Aggregator 1. (b) Composite Aggregator 2.

Figure 12: Runtime of the approximate solution w.r.t. δ.

Table 2: Approximation quality for composite aggregator F_1.

Cardinality $\lvert O \rvert$	δ			
	0.1	0.2	0.3	0.4
1×10^8	1.02819	1.02826	1.02829	1.02829
2×10^8	1.05659	1.05659	1.05678	1.05681



(a) Effect of the query rectangle. (b) Scalability.

Figure 13: Application to MaxRS problem.

is measured by $quality = \frac{d_{app}}{d_{opt}}$, where d_{app} is the distance of the region returned by the approximate solution and d_{opt} is the distance of the most similar region. We vary δ from 0.1 to 0.4, and report the quality of the approximate result for composite aggregator F_1 in Table 2. We observe that the approximation quality is quite good even when a large δ is used.

7.5 Application to the MaxRS Problem

We evaluate the proposed algorithm when it is adapted to solve the *MaxRS* problem. Recall that the *MaxRS* problem [21] aims to find a rectangle of given size that encloses the maximum number of spatial objects. The algorithm *Optimal Enclosure* (OE) with complexity of $O(n \log n)$ is the state-of-the-art solution to the *MaxRS* problem. As discussed in Section 2, the *MaxRS* problem is a special case of the *ASRS* problem. We can thus easily adapt the proposed *DS-Search* algorithm to solve the *MaxRS* problem. Specifically, as we are interested in the region that encloses the maximum number of objects, we estimate an upper bound for each cell instead of lower bound. The upper bound of a dirty cell g is the total number of rectangles that fully or partially cover g. Then we modify Algorithm 1 to greedily process the regions with the maximum upper bound (line 4 in Algorithm 1). We use *DS-Search* to denote the modified version of Algorithm 1 and compare it with *OE*.

Firstly, we evaluate effect of the size of the query rectangle. We randomly select 5×10^6 spatial objects from Tweet. We vary the rectangle size from q to $30q$. Fig 13a reports the runtime of the two algorithms w.r.t. the query rectangle size. We observe that *DS-Search* is about an order of magnitude faster than *OE*. Moreover, *DS-Search* is less sensitive to the size of the query rectangle.

We next evaluate the scalability. We vary the cardinality of the dataset from $1,000,000$ to $10,000,000$ spatial objects and report the runtime in Fig 13b. We observe that *DS-Search* scales well w.r.t. the number of spatial objects. It can finish in fewer than 20 seconds when there are 10,000,000 spatial objects, which enables it to handle real applications with massive data.

We conclude that *DS-Search* outperforms the state-of-the-art algorithm *OE* for the *MaxRS* problem, which is adopted in all subsequent studies [5, 11, 12, 24].

7.6 Case Study

In our case study, we run *DS-Search* on the 4,556 Foursquare POIs in Singapore. We adopt the composite aggregator $F = ((f_D, \text{Category}, \gamma_{all}))$ to compute the category distribution of the POIs in a region.

Fig 14(a) depicts three regions: "Orchard" (red), "Marina Bay" (black), and "Bugis" (blue). Specifically, "Orchard" is the query region, "Marina Bay" is the similar region discovered by *DS-Search*, and "Bugis" is a baseline region that is used to help us interpret

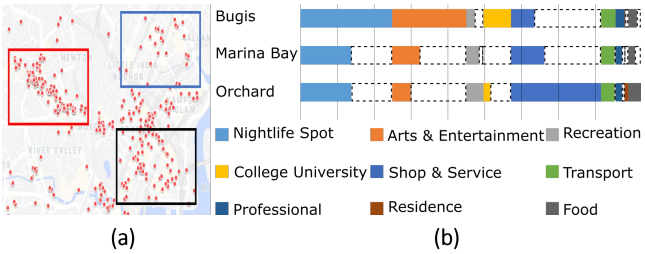


Figure 14: A Case study on Singapore.

Figure 15: "Orchard" is more related to "Marina Bay."

the result. The aggregate representations of the three regions are visualized as a stacked bar graph in Fig 14(b).

"Orchard" and "Marina Bay" are epicenters of shopping in Singapore that offer many luxury stores and entertainment options. As can be seen from Fig 14(b), most of the dimensions in the aggregate representations of the two regions are very similar. This explains why *DS-Search* thinks "Orchard" and "Marina Bay" are similar. In contrast, although "Bugis" and "Orchard" are similar in dimensions like "Food" and "Transport," they are quite different in the other dimensions, like nightlife spot and arts & entertainment. In fact, as shown in Fig 15, if we search the three regions in Google, there are more results about "Orchard" and "Marina Bay" than about "Orchard" and "Bugis," which indicates that "Marina Bay" is more similar to "Orchard" than "Bugis."

This case study shows that the *ASRS* functionality is useful for capturing a region's characteristics and for retrieving similar regions. Therefore, our proposed solution can be a useful tool in a range of real life applications. For instance, if a user enjoys exploring "Orchard," the proposed solution can be utilized to identify the similar region "Marina Bay" and recommend it to the user for further exploration.

8. CONCLUSIONS

The need for advanced retrieval of regions with selected characteristics has gained in prominence due to the availability of increasingly massive volumes of geo-tagged data. We define so-called composite aggregators that are able to capture a region's characteristics. We then formalize and study a new problem, the attribute-aware similar region search problem. To this end, we propose a novel algorithm called the *DS-Search* algorithm. We also propose indexing along with pruning techniques to improve the efficiency of *DS-Search*. Since approximate answers are acceptable in many applications, we extend *DS-Search* to find regions that are similar to the exact solution with error bounds. The experimental study shows that *DS-Search* is 2–3 orders of magnitude faster than a baseline algorithm adapted from the sweep-line algorithm. We also show that we can extend *DS-Search* to address the *MaxRS* problem with slight modifications while outperforming the baseline algorithm by one order of magnitude. As part of future work, we intend to take the inner structure of the region, i.e., the spatial distribution of the objects into consideration to measure the similarity between regions.

Acknowledgments. This research is supported by a MOE Tier-2 grant MOE2016-T2-1-137, and a MOE Tier-1 grant RG31/17.

9. REFERENCES

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[2] X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi. Efficient processing of spatial group keyword queries. *ACM Transactions on Database Systems*, 40(2):13, 2015.

[3] A. Cary, O. Wolfson, and N. Rishe. Efficient and scalable method for processing top-k spatial Boolean queries. In *Proceedings of the International Conference on Scientific and Statistical Database Management*, pages 87–95, 2010.

[4] H.-J. Cho and C.-W. Chung. Indexing range sum queries in spatio-temporal databases. *Information and Software Technology*, 49(4):324–331, 2007.

[5] D.-W. Choi, C.-W. Chung, and Y. Tao. A scalable algorithm for maximizing range sum in spatial databases. *PVLDB*, 5(11):1088–1099, 2012.

[6] D.-W. Choi, J. Pei, and X. Lin. Finding the minimum spatial keyword cover. In *Proceedings of the 32nd IEEE International Conference on Data Engineering*, pages 685–696, 2016.

[7] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: efficient geo-search query processing. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 423–432, 2011.

[8] G. Cong, K. Feng, and K. Zhao. Querying and mining geo-textual data for exploration: Challenges and opportunities. In *Proceedings of the 32nd IEEE International Conference on Data Engineering Workshops*, pages 165–168. 2016.

[9] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[10] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 656–665, 2008.

[11] K. Feng, G. Cong, S. S. Bhowmick, W.-C. Peng, and C. Miao. Towards best region search for data exploration. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1055–1070, 2016.

[12] K. Feng, T. Guo, G. Cong, S. S. Bhowmicks, and S. Ma. Surge: Continuous detection of bursty regions over a stream of spatial objects. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 1292–1295, 2018.

[13] T. Guo, X. Cao, and G. Cong. Efficient algorithms for answering the m-closest keywords query. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 405–418. 2015.

[14] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *Proceedings of the 19th International Conference on Scientific and Statistical Database Management*, article 16, 2007.

[15] M. Jurgens and H.-J. Lenz. The Ra*-tree: an improved R*-tree with materialized data for supporting range queries on OLAP-data. In *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, pages 186–191. 1998.

[16] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *Proceedings of the 2015 International Conference on Management of Data*, volume 30, pages 401–412, 2001.

[17] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng. On trip planning queries in spatial databases. In *Proceedings of the International Symposium on Spatial and Temporal Databases*, pages 273–290. Springer, 2005.

[18] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. IR-Tree: An efficient index for geographic document search. *IEEE Transactions on Knowledge and Data Engineering*, 23(4):585–599, 2011.

[19] X. Ma, S. Shekhar, H. Xiong, and P. Zhang. Exploiting a page-level upper bound for multi-type nearest neighbor queries. In *Proceedings of the 14th annual ACM International Symposium on Advances in Geographic Information Systems*, pages 179–186, 2006.

[20] M. I. Mostafiz, S. Mahmud, M. M.-u. Hussain, M. E. Ali, and G. Trajcevski. Class-based conditional MaxRS query in spatial data streams. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, Article 13. 2017.

[21] S. C. Nandy and B. B. Bhattacharya. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Computers & Mathematics with Applications*, 29(8):45–61, 1995.

[22] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP operations in spatial data warehouses. In *Proceedings of the International Symposium on Spatial and Temporal Databases*, pages 443–459. Springer, 2001.

[23] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. *The VLDB Journal*, 17(4):765–787, 2008.

[24] Y. Tao, X. Hu, D.-W. Choi, and C.-W. Chung. Approximate maxrs in spatial databases. *PVLDB*, 6(13):1546–1557, 2013.

[25] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 688–699, 2009.